Stay Fresh: Speculative Synchronization for Fast Distributed Machine Learning

Chengliang Zhang†, Huangshi Tian†, Wei Wang†, Feng Yan‡

†Hong Kong University of Science and Technology

‡University of Nevada, Reno

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

THE DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING 計算機科學及工程學系

Outline

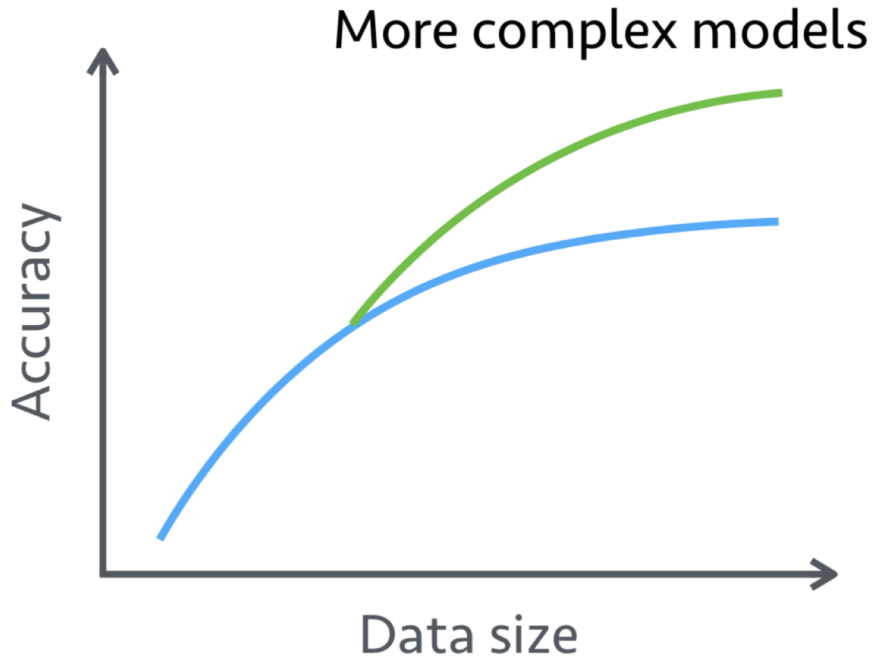
- Background and Motivation
- Insights of Distributed Asynchronous Learning
- Solution: Speculative Synchronization
- Implementation
- Evaluation
- Conclusion

Large Scale Machine Learning

- Machine learning learns from data
- More data leads to better accuracy
- Complex models can further improve accuracy

Big data and complex models

Distribute workload among many machines



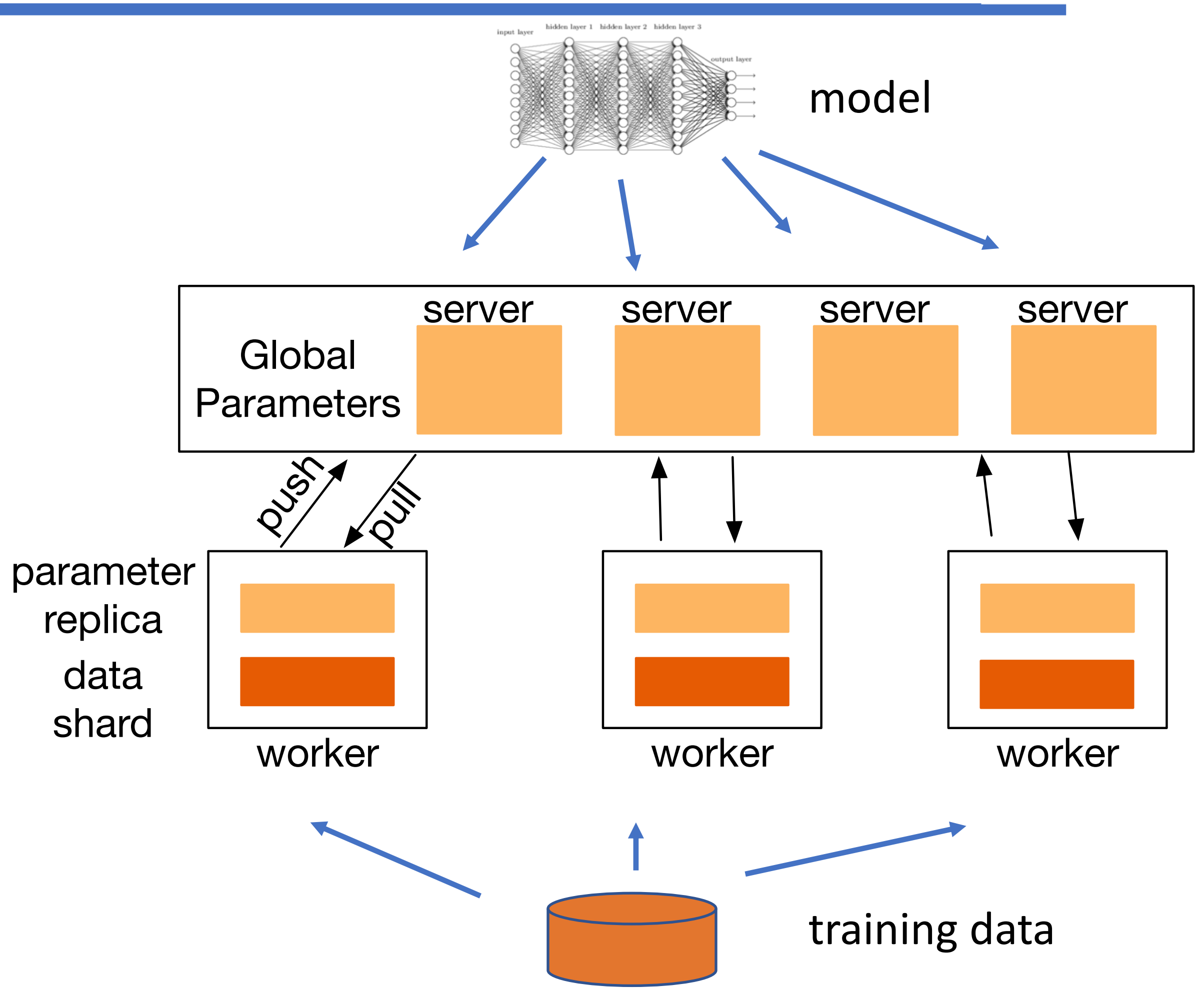
[1] Li, Mu. "Scaling distributed machine learning with system and algorithm co-design." Diss. Intel, 2017.

Parameter Server

state-of-the-art architecture for distributed ML

Iterate until stop:

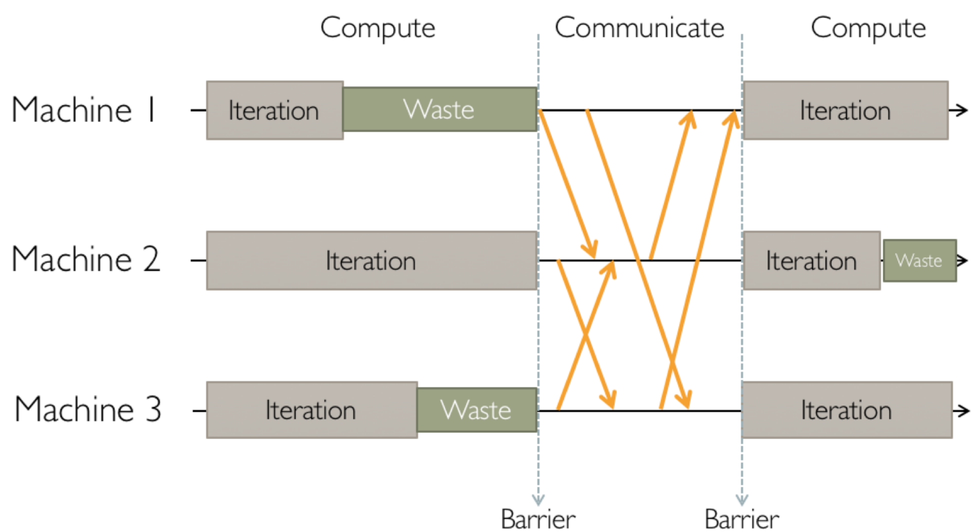
- workers compute updates
- workers ***push*** updates
- servers update model
- workers ***pull*** updated model



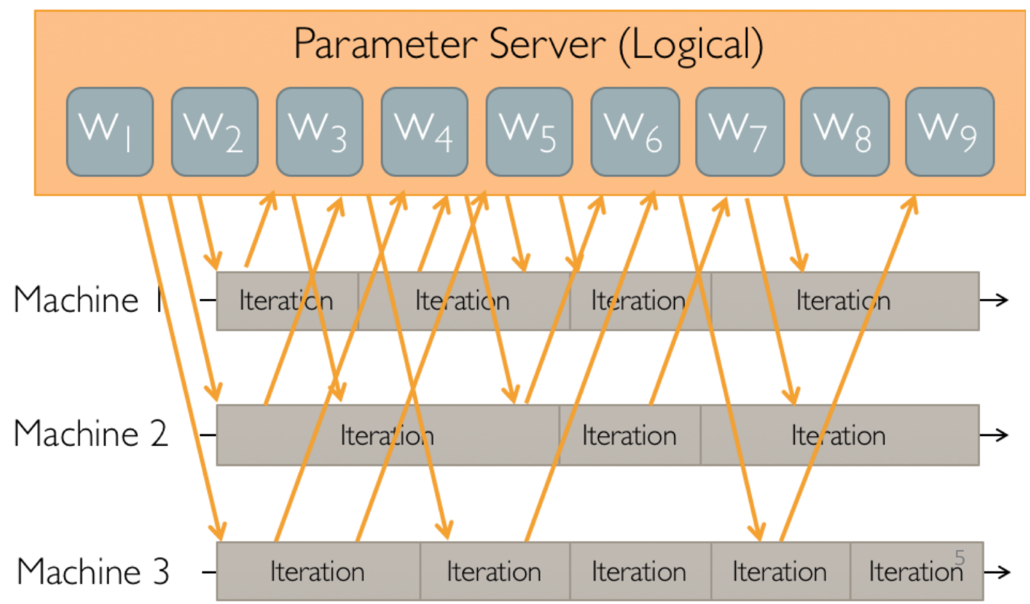
[1] Li, Mu, et al. "Scaling Distributed Machine Learning with the Parameter Server." OSDI. Vol. 14. 2014.

Synchronization Schemes

- Bulk Synchronous Parallel (BSP)
  - Strong consistency
  - Straggler
  - Concurrent communication
  - Low throughput



- Asynchronous Parallel (ASP)
  - No barrier
  - High throughput
  - Cheap synchronization
  - Inconsistency



Inconsistency and Convergence

- Inconsistent model replicas among workers
- Stale parameters poison convergence
- Stale Synchronous Parallel (SSP) : bound the staleness

Parameter replica:
the fresher the better

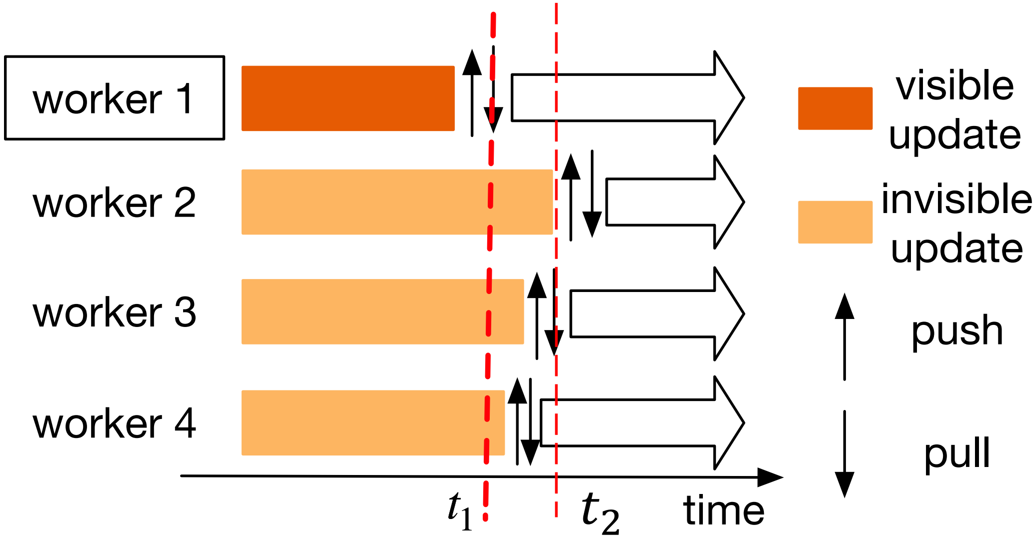
- tradeoff between update rates and update quality

Asynchronous learning → Higher rate of updates — Good for convergence

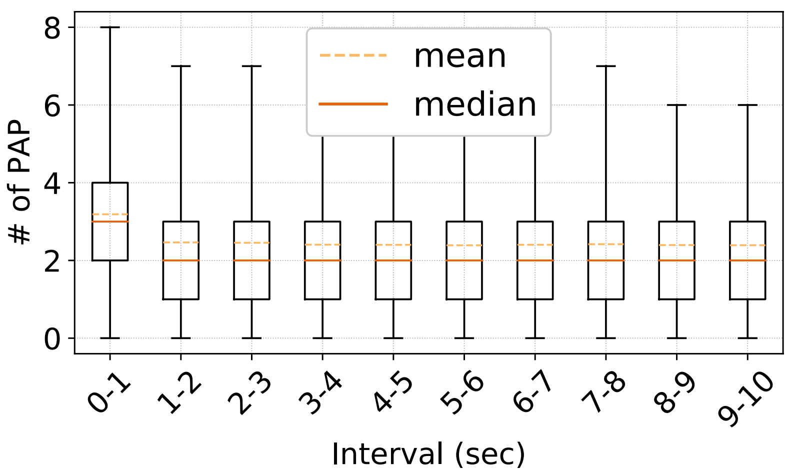
Asynchronous learning → Lower quality of updates — Bad for convergence

[1] J. Langford, A. J. Smola, and M. Zinkevich, “Slow learners are fast,” in NIPS, 2009.

Insights: Pushes after Pull

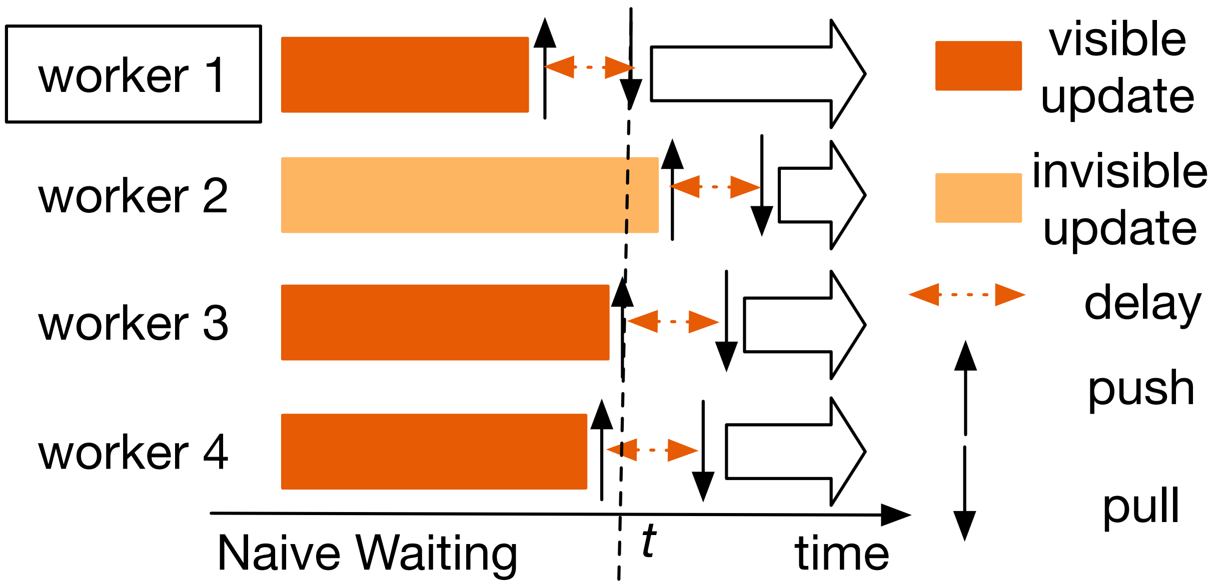


- Worker 1 eagerly pulls after push
- Misses updates from others



- 3 PAPs on average
- Missed opportunity for fresher parameters

Naïve Waiting

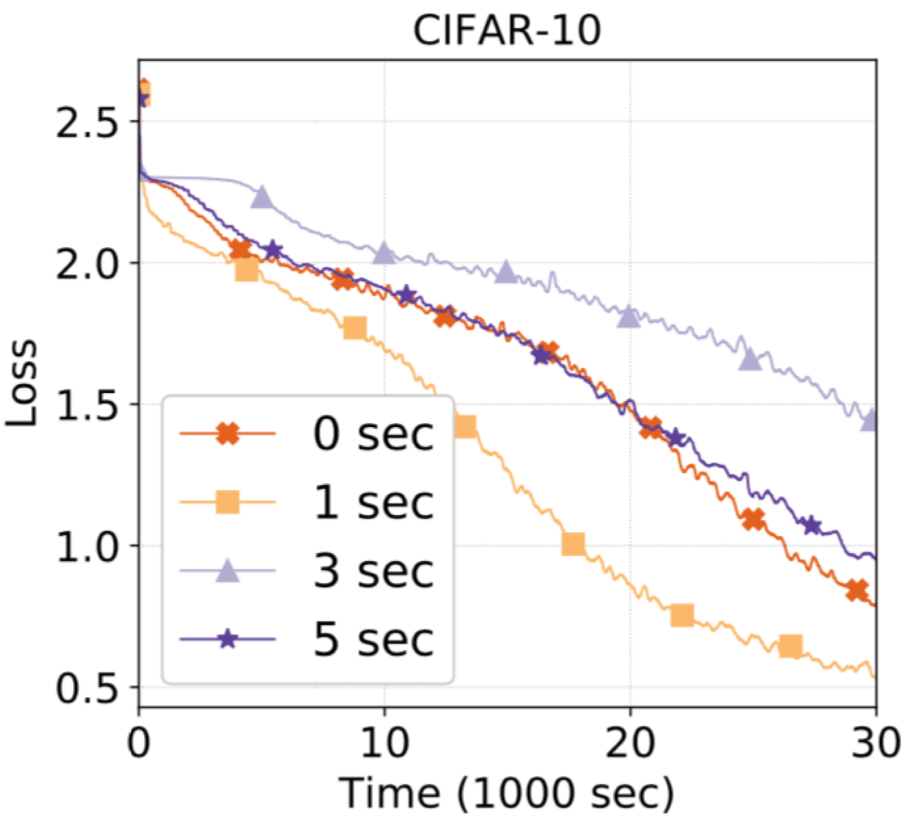


Intuition

simply *defer* the pull request

PAPs will be included

Naïve Waiting



- Works, but not always

Desired:

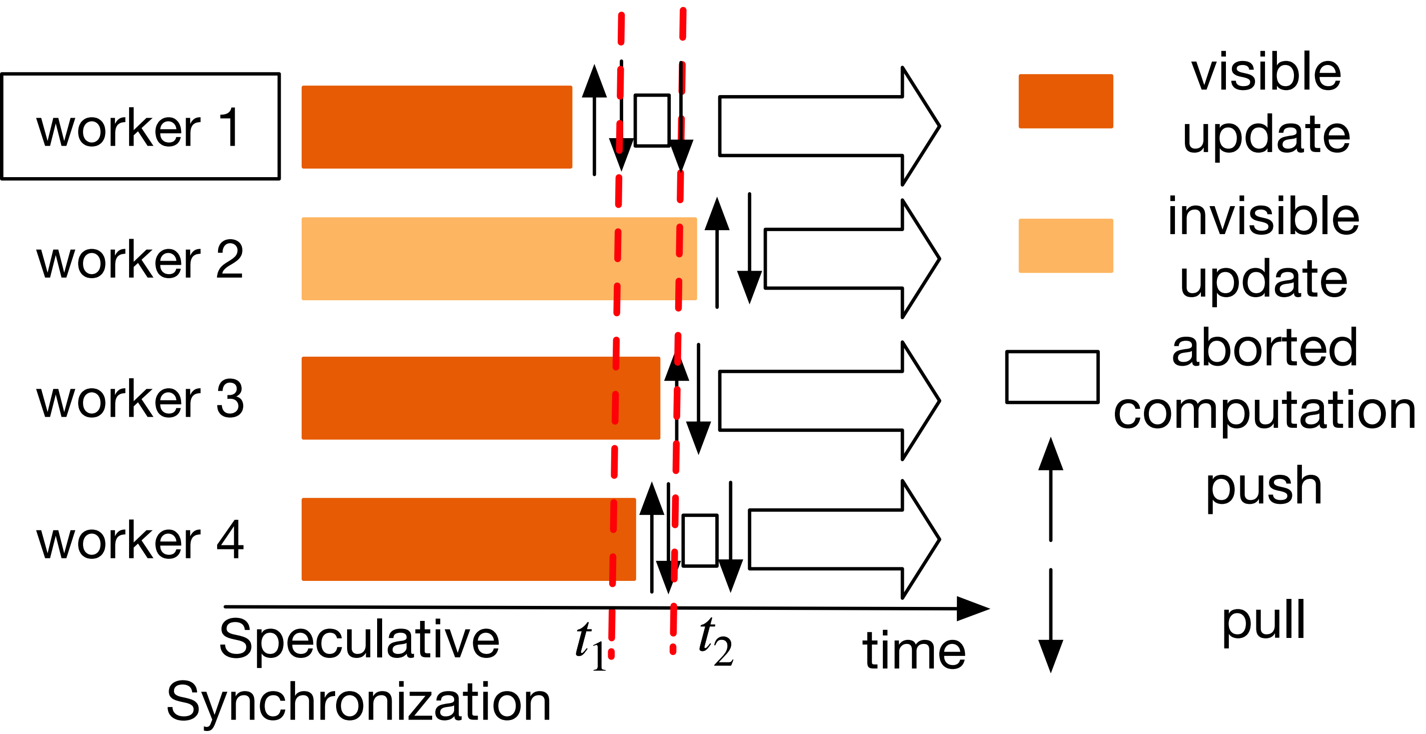
freshness gain > computation loss

Invalid wait:

freshness gain < computation loss

Speculative Synchronization

SpecSync: speculatively abort the ongoing computation and start over with fresher parameters



Gain:
fresh parameters

Lost:
aborted computation

Speculative Synchronization

Advantages:

- Avoid invalid waits
- Minimize the cost of wasted computing cycles
- Suitable for asynchronous models including ASP and SSP

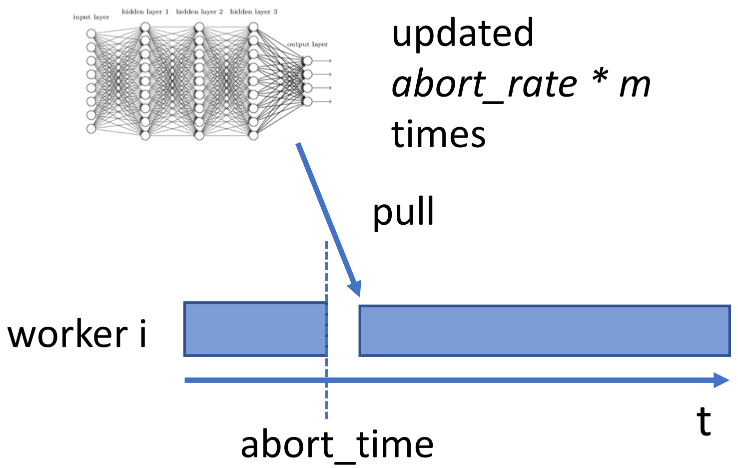
Challenges:

- Efficient communication
  - Exchange worker progress
  - Additional parameter pull
- When to abort and restart

Hyperparameters

abort_time and abort_rate

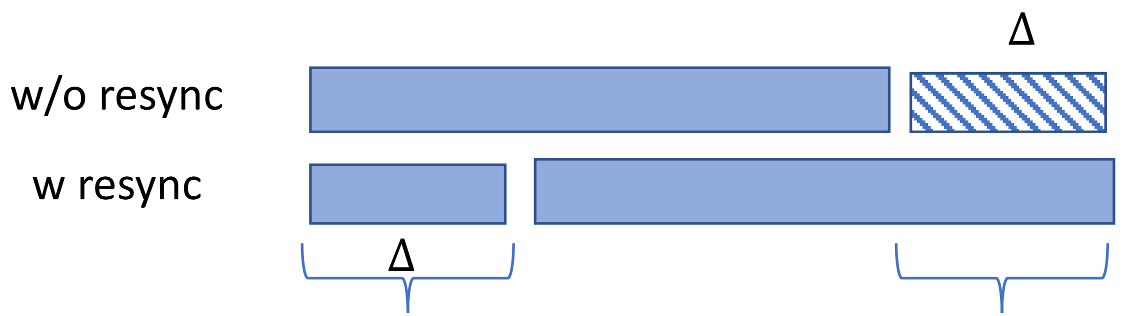
For a worker, in the first *abort_time*, if more than *abort_rate * m* updates arrive at severs, re-synchronize.



Given a workload, how do we choose abort_time and abort_rate?

Formulation

How to model the gain and loss of re-synchronization?



Gain:
More updates from other workers

Loss:
Other worker lose 1 update from the delay

net gain = *uncovered updates* - *missed peers*

$$F_{i,\tau}(\Delta) = u_{i,\tau}(\Delta) - l_{i,\tau}(\Delta)$$

Only re-sync when $F_{i,\tau}(\Delta) > 0$

Formulation

Sum up the gain over all workers in epoch τ

$$maximize_{\Delta} F_{\tau}(\Delta) = \sum_{i=1}^{m} (u_{i,\tau}(\Delta) - l_{i,\tau}(\Delta))$$

How to solve?

- Direct solution: require exact push/pull sequence ❌
- Estimation: use traces and expectations from last epoch

Adaptive Tuning

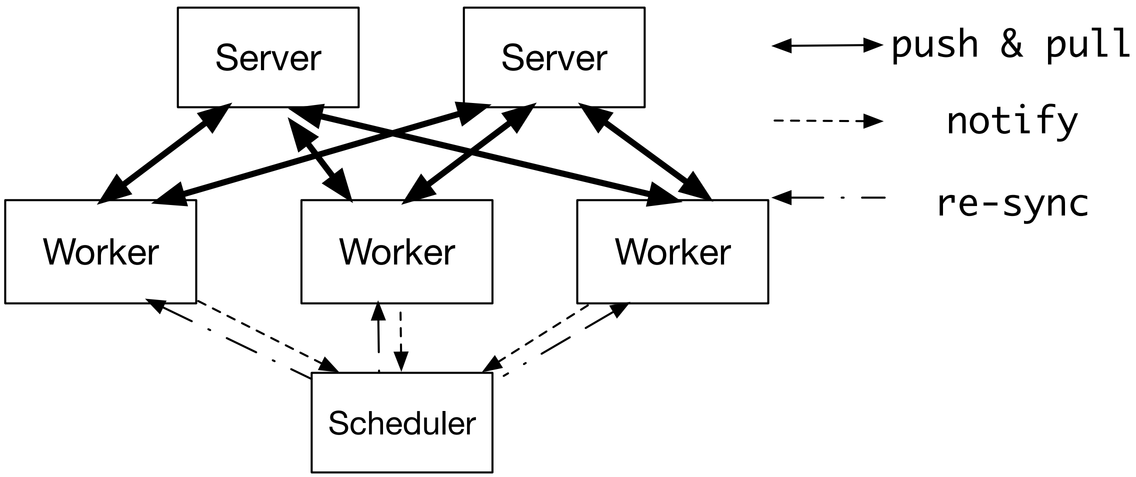
Once we have optimal Δ^*

- Set *abort_time* to Δ^* to maximize potential gain
- Set *abort_rate* to the expected missed peers
- Only abort if the gain outweighs loss

Implementation

An extension to MXNet.

Centralized design



Scheduler:

- Keep tracks of updates
- Tune abort_time and abort_rate
- Issue re-sync command to workers

Evaluation

- Effectiveness
  - Accuracy and runtime
- Robustness
  - heterogeneity and scalability
- Communication Overhead

Evaluation Setup

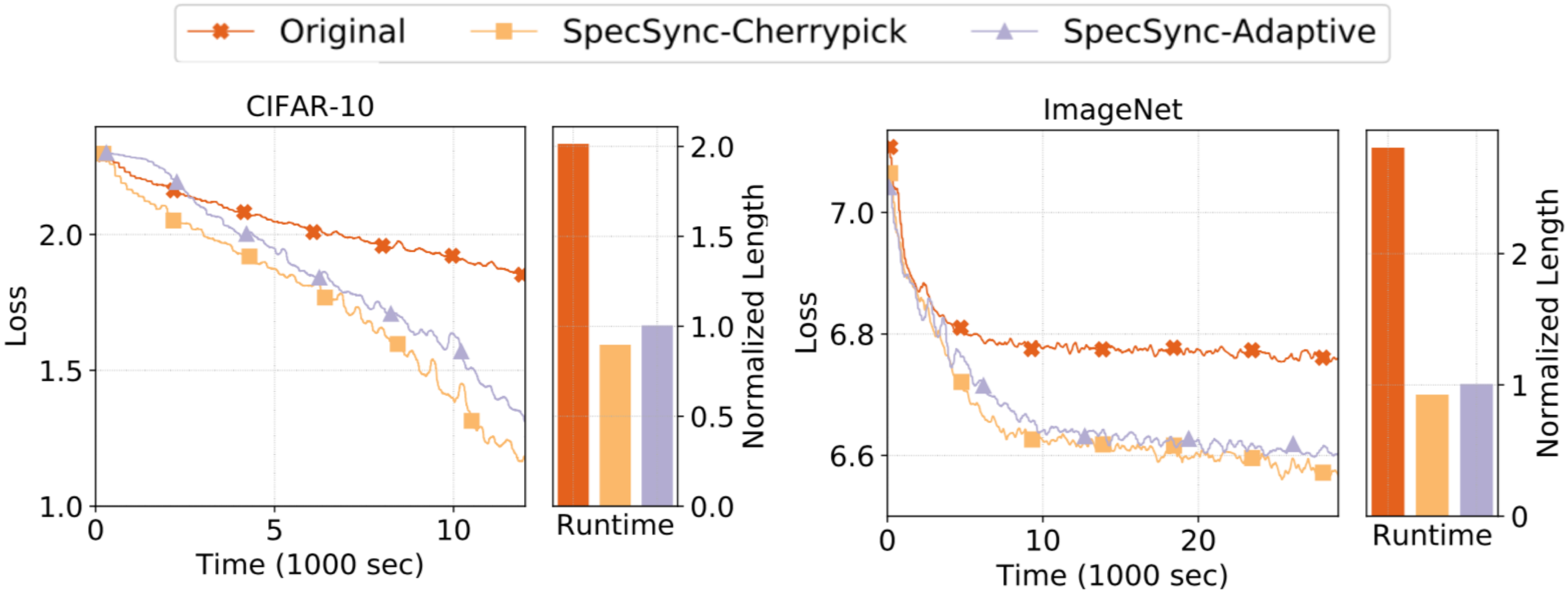
- Workload

workload	# parameters	dataset	dataset size
MF	4.2 million	Movielens	100,000
CIFAR-10	2.5 million	CIFAR-10	50,000
ImageNet	5.9 million	ImageNet	281,167

- Schemes
  - Original: stock MXNet asynchronous implementation
  - SpecSync-*cherrypick*: SpecSync with cherrypicked hyperparameters
  - SpecSync-*adaptive*: SpecSync with adaptively tuned hyperparameters
- Testbed
  - AWS EC2

Effectiveness

40 m4.xlarge instances

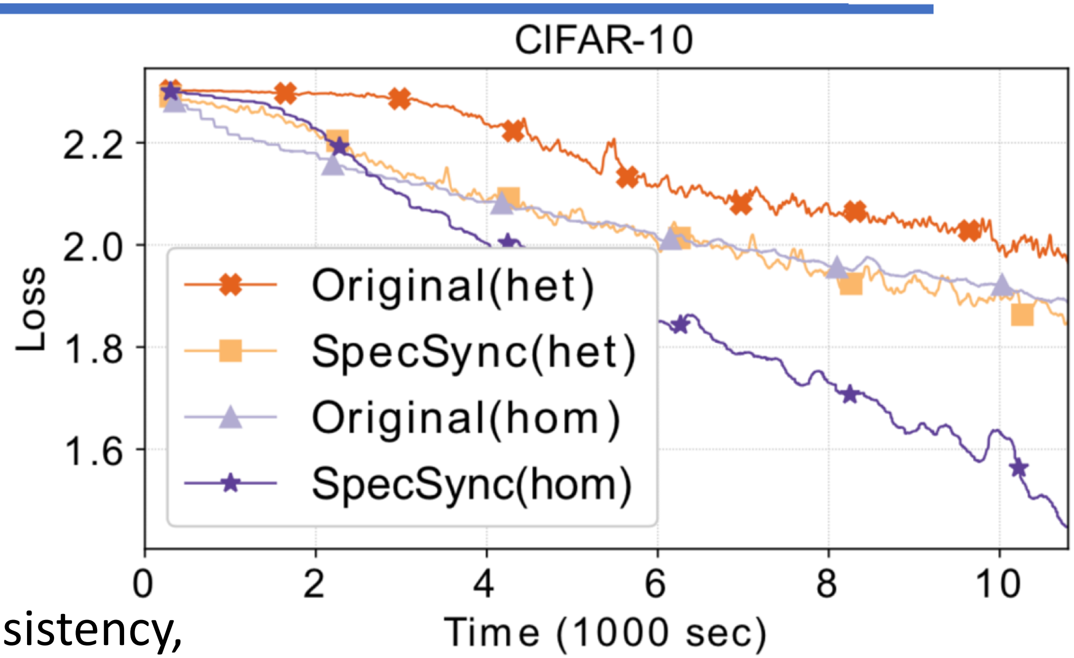


- SpecSync improves performance
- 2.97× 2.25× 3× speedup respectively
- Adaptive tuning, comparable speedups

Robustness

- Heterogeneity

10 m3.xlarge+ 10 m3.2xlarge +
10 m4.xlarge + 10 m4.2xlarge

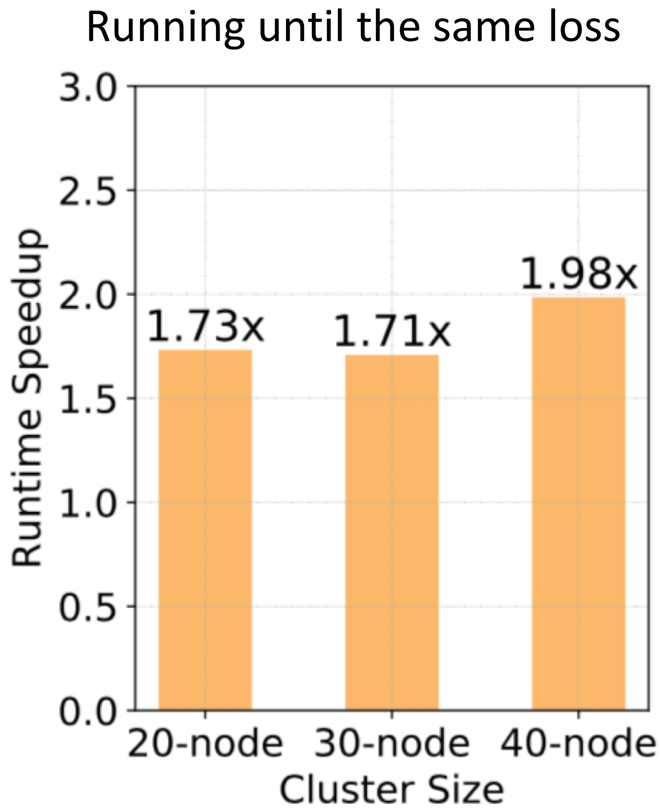


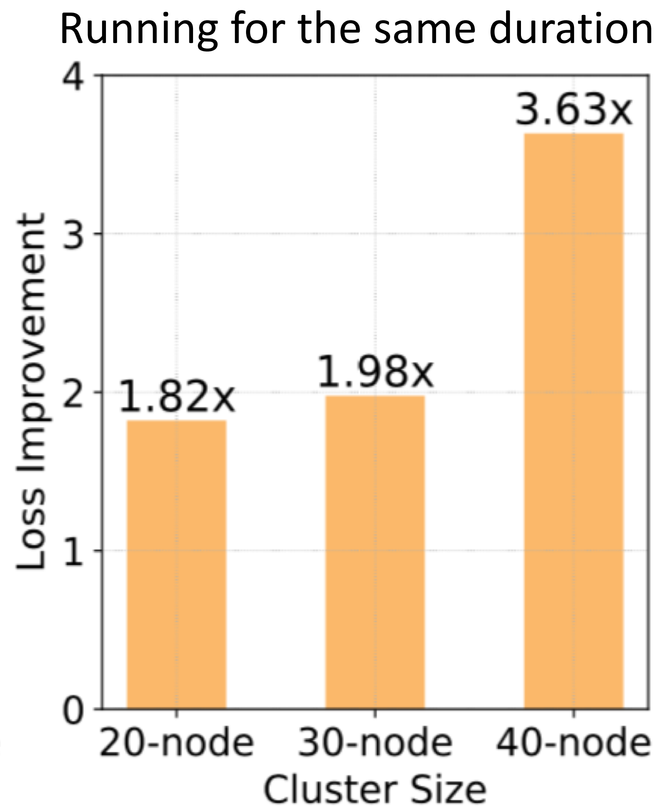
- Heterogeneity increases inconsistency, affects performance
- SpecSync work both in homogeneous and heterogeneous settings

Robustness

- Scalability

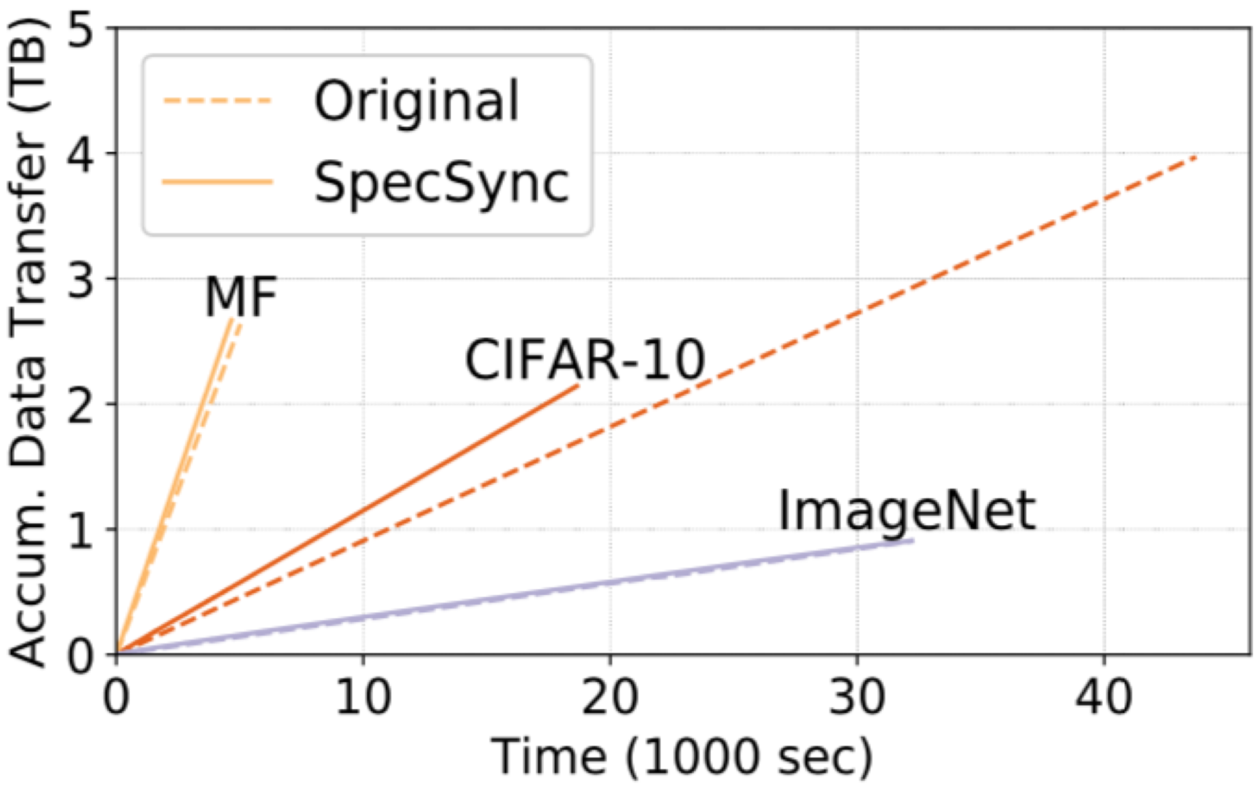
20, 30, 40 m4.xlarge





Communication Overhead

SpecSync introduces additional communication



- The accumulated communication does not increase

Conclusion

- Investigated inconsistency in distributed ML
- Proposed SpecSync to actively improve freshness
- Designed an adaptive hyperparameter tuning algorithm
- Implemented SpecSync atop MXNet and evaluated it.

Thank you for listening!

Q&A